Timing Studies for Efficient Data Transfer Between the NOvA Data Concentrator Modules and the NOvA Trigger Processors

Stephen Foulkes
Fermi National Accelerator Laboratory

Abstract
This document describes performance testing and results for the NOvA data acquisition development effort. The tests attempt to benchmark the performance of the Linux TCP/IP stack in an effort to optimize the software and hardware for the NOvA trigger processors.

1 Test System Setup

Four nodes in the FCC 2 computer room were used for this test. Each was a dual AMD Athlon 1900+ with 1GB of ram. The tests made use of the 3com 3c905c 10/100 Ethernet cards that were integrated with the motherboard. Each card had an MTU of 1500 bytes and was plugged into the same 10/100Mb switch.

All systems were identical, running Fermi Linux 4.2 with a 2.6.9 SMP kernel. The test software was compiled with the GNU C++ compiler, version 3.4.4. The only modification made to the test nodes was to open a port in the firewall so that test nodes could receive data.

2 Test System Software

The times() function was used to determine the run time of the program as well as the amount of processor time used. This is the same method that the "time" utility uses.

2.1 Data Concentrator Module

The software that simulated the Data Concentrator Module would open up a series of connections to a buffer node and then randomly select one of the open connections and send a stream of data. The number of connections, total number of transmissions, amount of data sent per transmission and the rate of transmissions are specified on the command line.

2.2 Trigger Processor Software

The Trigger Processor software listens for and then accepts connections from the Data Concentrator Module. The Trigger Processor waits for the Data Concentrator Module to send data on a connection, and then copies the data to a buffer. The Trigger Processor software is instrumented to keep track of the total run time, the total CPU time used, the number of iterations in the main loop, as well as the number of connections that are handled in each iteration of the main loop. The total number of transmission, total number of connections and the size of each transmission are passed to the Trigger Processor software from the command line.

The Trigger Processor software is a single threaded process, and uses the epoll interface available in the Linux 2.6 kernel to manage its open connections. Epoll is very similar to select and poll, in which it monitors a number of connections and then determines which connections are able to be read from or written to. What makes epoll better than select() or poll() is that the mechanism for passing down the list of connections to monitor is separate from the mechanism used to get the state of each connection. With epoll the

connection list does not have to be passed to the kernel on every iteration of the main loop.

2.3 Python Scripts

Two python scripts were written, one to control the Data Concentrator Module and one to control the Trigger Processor. A range of options would be specified in each script, and the script would take care of iterating through each option and logging the results.

3 Tests Run

Two series of tests were run, one with ~1900 transmissions per second and one with ~3800 transmissions per second. Each set of parameters was repeated four times during each test. In each test the number of open connections varied from 250 to 1000 in increments of 50. Six transmission sizes consisting of 10 bytes, 100 bytes, 250 bytes, 500 bytes, 1000 bytes and 2000 bytes were run for each set of open connections. A transmission size of 4000 bytes was also run for the test with ~1900 transmissions per second. This could not be run for the other test due to the fact that only 100Mb equipment was used.

4 Test Results

4.1 The number of open connections has little impact on CPU usage

Figure 1 plots the number of open connections vs. CPU usage for the test run with ~1900 transmissions per second and figure 2 plots the same for the test run with ~3800 transmissions per second.

All tests run with transmission sizes of 500 and 1000 bytes are completely flat across the range of open connections. Tests run with smaller transmissions actually perform better with a larger number of connections.

The 10, 100 and 250 bytes tests experienced a factor of 3 less CPU usage with 1000 open connections then with 250 open connections. Figures 3 and 4 plot the CPU usage vs. the number of connections handled per loop iteration. These show that CPU usage increases as the number of connections handed per loop iteration decreases. This makes sense, as less loop iterations means less system calls and jumps between user and kernel space.

4.2 Jumbo Packets will go a long way to increasing transfer efficiency

All tests were run on 100Mb hardware with a MTU of 1500 bytes, so Jumbo Packets were not directly tested. The effects that Jumbo Packets will have can be inferred by comparing the CPU usage between the tests with transmission sizes of 500 and 1000 bytes and tests run with transmission sizes of 2000 and 4000 bytes. The larger transmissions have to be broken up into multiple packets, resulting in more overheard all around.

The increased CPU usage can be seen on figures 1 and 2. The reduction in the efficiency of the main loop can be seen for the larger transmissions sizes in figures 3 and 4. The main loop would only handle one half and one third of a transmission per loop iteration for the larger transmissions.

The tests behaved as expected. The TCP is a stream protocol, in that it will only guarantee that data arrives in order and that it is not corrupted. It does not guarantee that message boundaries will be kept intact. The Linux TCP/IP implementation limits the TCP MSS to that of the MTU of the link layer, in this case 100Mb Ethernet. Data is split up into 1500 byte packets. We would expect to see at most 1500 bytes received on each iteration of the loop, which is what the tests found.

5 Figures

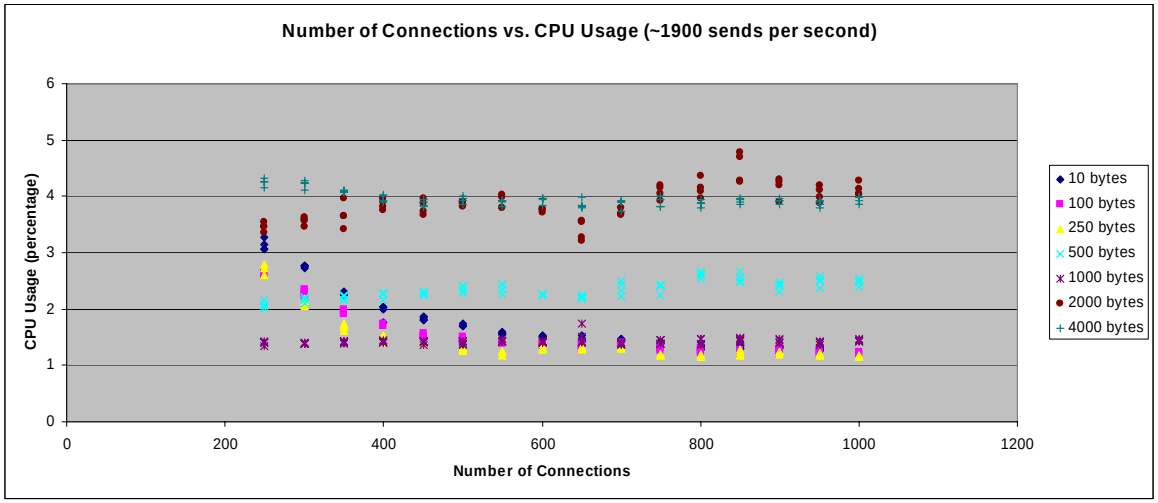

Figure 1

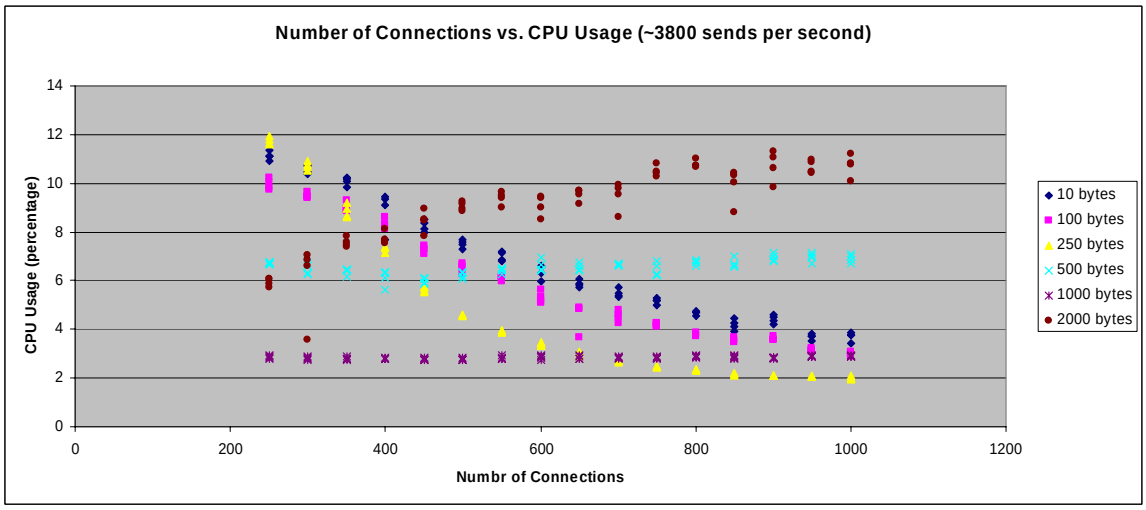

Figure 2

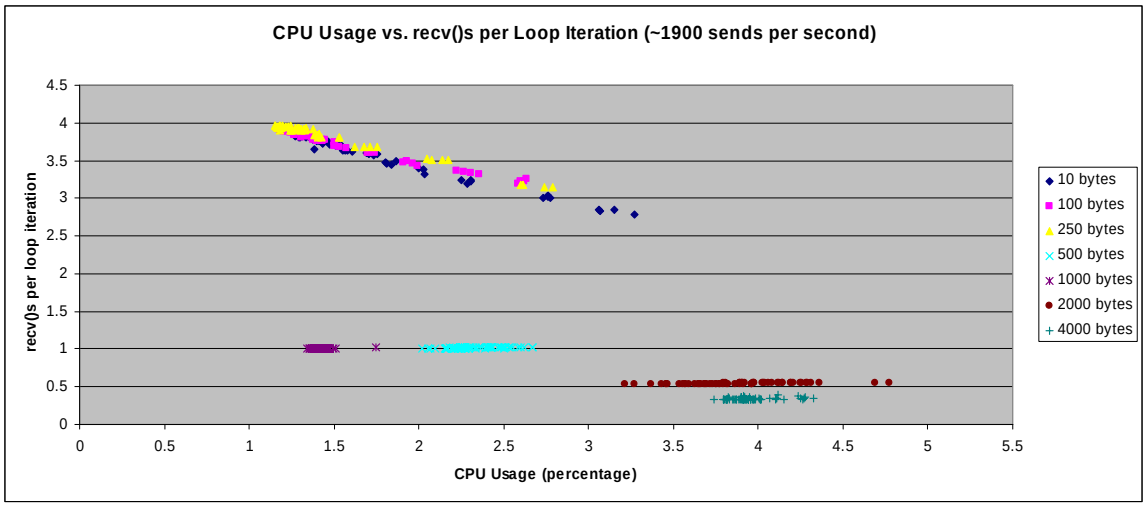

Figure 3

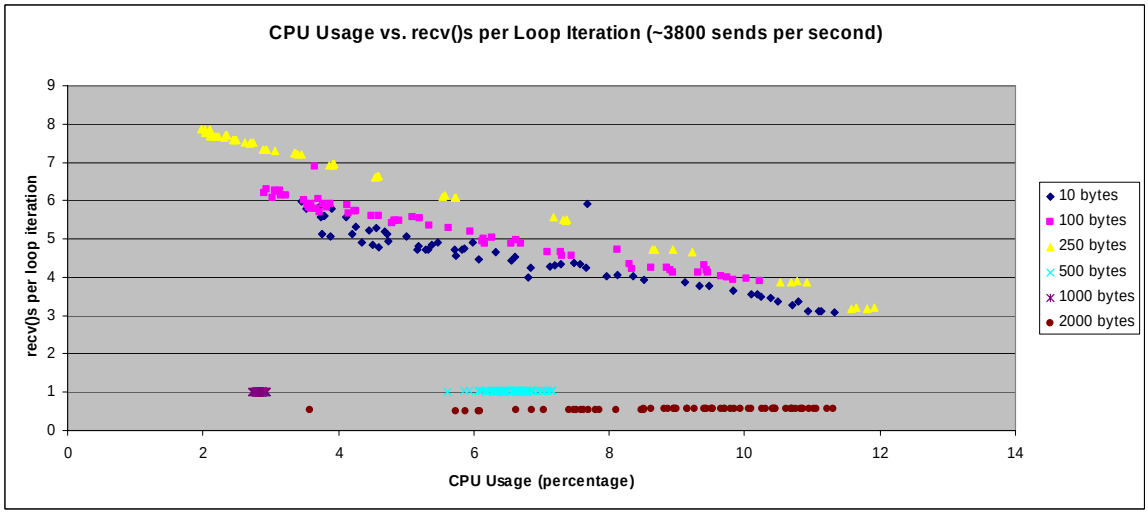

Figure 4